Neural Network Initialization

Greg Anderson

February 22, 2022

In this supplement, we will discuss how to initialize the weights in neural networks. We saw in class that setting all the weights to zero does not work because the gradients will always be zero, and we said that usually we will randomly choose weight values from a normal distribution. Further, we said that the mean of the normal distribution should be zero. Here, we are interested in determining the standard deviation we should use in our normal distribution when initialization network weights.

1 Setup

First, let's introduce a property of normal distributions which will be useful in our analysis. If $\vec{a}$ is any vector sampled from a multivariate normal distribution $\vec{a} \sim \mathcal{N}(\vec{\mu}, \Sigma)$ and $\vec{x}$ is any vector, then

$$\vec{a}^T\vec{x} \sim \mathcal{N}\left(\vec{\mu}^T\vec{x}, \vec{x}^T\Sigma\vec{x}\right)$$

For our purposes, we will be assuming that $\vec{\mu} = 0$ and $\Sigma = \sigma^2 I$ for some σ (where I is the identity matrix of the appropriate dimension). That is, each element of $\vec{a}$ is sampled independently from the (univariate) normal distribution $\mathcal{N}(0, \sigma^2)$. In that case, we have that $\vec{\mu}^T\vec{x} = 0$ and

$$\vec{x}^T\Sigma\vec{x} = \vec{x}^T(\sigma^2 I)\vec{x} = \sigma^2\vec{x}^T\vec{x} = \sigma^2 \left\|\vec{x}\right\|^2 .$$

Therefore, $\vec{a} \sim \mathcal{N}(0, \sigma^2 I)$ and $\vec{a}^T\vec{x} \sim \mathcal{N}(0, \|x\|^2\sigma^2)$.

Since we are often going to talk about matrix-vector multiplications in neural networks, let's extend the previous result to handle matrices. When we do a matrix-vector multiplication, what we are really doing is computing a dot product between each row of the matrix and the vector. That is, for a matrix A and a vector $\vec{x}$, we have $(A\vec{x})_i = A_i\vec{x}$. where A_i is the i'th row of A. If we assume each element of A is independently sampled from a normal distribution $\mathcal{N}(0, \sigma^2)$, then we have $A_i\vec{x} \sim \mathcal{N}(0, \|x\|^2\sigma^2)$ for each i, and therefore $A\vec{x} \sim \mathcal{N}(0, \|x\|^2\sigma^2 I)$.

Now that we can talk about how random matrix multiplication affects a vector, let's look at how a ReLU affects a random vector. Suppose $\vec{x} \sim \mathcal{N}(0, \sigma^2 I)$. We are interested in finding the expected size of the output vector after a ReLU. That is, we want to find $\mathrm{E}[\|\mathrm{ReLU}(\vec{x})\|^2]$. The key observation here is that because the normal distribution is symmetrical and because the mean is assumed to be zero, in expectation half of the elements of $\vec{x}$ will be less than zero. Let n_x be the number of elements in $\vec{x}$. Then we have

$$\mathrm{E}\left[\|\mathrm{ReLU}(\vec{x})\|^2\right] = \mathrm{E}\left[\sum_{i=1}^{n_x} \max\left(\vec{x}_i, 0\right)^2\right] = \sum_{i=1}^{n_x} \mathrm{E}\left[\max\left(\vec{x}_i, 0\right)^2\right]$$

Because $\vec{x}_i \sim \mathcal{N}(0, \sigma^2)$, we know that $\mathrm{E}[\vec{x}_i^2] = \sigma^2$ because this is a property of the normal distribution. Any of the values which are below zero will be set to zero by the maximum, so we end up with $\mathrm{E}\left[\max\left(\vec{x}_i, 0\right)^2\right] = \sigma^2/2$ and therefore

$$\mathrm{E}\left[\|\mathrm{ReLU}(\vec{x})\|^2\right] = \sum_{i=1}^{n_x} \mathrm{E}\left[\max\left(\vec{x}_i, 0\right)^2\right] = \frac{1}{2} n_x \sigma^2.$$

2 Network Analysis

Let's look at what happens when we use the relationships we discussed above to analyze the values passed through a neural network. To keep things a little simpler, we will assume that our network only contains fully connected linear layers and ReLUs, and we will further assume that all the biases are zero. That is, our network is defined by the following equations, where $\vec{x}$ is the input and W_i is the weight of the i'th layer.

$$\begin{aligned}
\vec{b}_0 &= \vec{x} \\
\vec{a}_{i+1} &= W_{i+1}\vec{b}_i \\
\vec{b}_{i+1} &= \mathrm{ReLU}(\vec{a}_{i+1})
\end{aligned}$$

Here $\vec{a}_i$ is the output of the i'th linear layer and $\vec{b}_i$ is the output of the ReLU whose input is $\vec{a}_i$.

Now let's assume each layer is initialized with each weight value drawn independently from $\mathcal{N}(0, \sigma_{W_i}^2)$. Notice that the standard deviation may be different for differen layers, but within a single layer each element of the weight matrix is drawn from the same distribution. We will also use n_i to indicate the number of elements in $\vec{b}_i$. After the first layer, we have $\vec{a}_1 = W_1\vec{b}_0$ and therefore

$$\vec{a}_1 \sim \mathcal{N}(0, \|\vec{x}\|^2\sigma_{W_1}^2 I).$$

Let $\sigma_1 = \|\vec{x}\|\sigma_{W_1}$ so that $\vec{a}_1 \sim \mathcal{N}(0, \sigma_1^2)$. Now after we pass through the ReLU, we have

$$\mathrm{E}\left[\left\|\vec{b}_1\right\|^2\right] = \frac{1}{2} n_1 \sigma_1^2.$$

We are going to make an additional simplifying assumption here which is that $\|\vec{b}_i\|^2 \approx \mathrm{E}[\|\vec{b}_i\|^2]$. Continuing inductively: for $i > 1$, we have $\vec{a}_i = W_i\vec{b}_{i-1}$ so

$$\vec{a}_i \sim \mathcal{N}(0, \|\vec{b}_{i-1}\|^2\sigma_{W_i}^2 I).$$

Similar to the first layer, let $\sigma_i = \|\vec{b}_{i-1}\|\sigma_{W_i}$ so that $\vec{a}_i \sim \mathcal{N}(0, \sigma_i^2)$. From here, we find

$$\left\|\vec{b}_i\right\|^2 \approx \mathrm{E}\left[\left\|\vec{b}_i\right\|^2\right] = \frac{1}{2} n_i \sigma_i^2.$$

Now let's combine some results: since $\sigma_i = \|\vec{b}_{i-1}\|\sigma_{W_i}$ and $\|\vec{b}_{i-1}\|^2 = n_{i-1}\sigma_{i-1}^2/2$ we have that for $i > 1$

$$\sigma_i = \frac{1}{\sqrt{2}} \sigma_{W_i} \sigma_{i-1} \sqrt{n_{i-1}}.$$

This gives us a recurrance relation which can tell us the expected magnitude of the output of each layer based on the output of the previous layer. If we apply this relationship repeatedly, we can represent the magnitude of an output by the following expression:

$$\sigma_i = \|\vec{x}\|^2 \sigma_{W_1} \prod_{k=2}^{i} \frac{1}{\sqrt{2}} \sigma_{W_k} \sqrt{n_{k-1}}.$$

This result gives us an expression to compute the size of the output of any layer in a randomly initialized network, depdending on the input size and the initialization parameters of all preceding layers.

We can go through a very similar process to compute the magnitude of the gradients in a random network given the magnitude of the gradient with respect to the logits. Because the process is almost exactly the same, I will spare you the details and just present the final result:

$$\hat{\sigma}_i = \|\nabla_{\vec{o}} \ell(\vec{o})\| \, \sigma_{W_N} \prod_{k=i}^{N-1} \frac{1}{\sqrt{2}} \sigma_{W_k} \sqrt{n_i}$$

In order to train the network effectively, we want to choose σ_{W_i} such that the output and gradient magnitudes are roughly the same for all layers of the network. To do this, we want each term in the products above to be one. This leads to our two main initialization schemes: Kaiming initialization and Xavier initialization.

In Kaiming initialization, we choose either outputs or gradients, and try to keep the magnitudes of the chosen values constant. For example, if we pick outputs, then we want σ_i to be the same for all i. To do this, we set

$$\sigma_{W_i} = \sqrt{2} \frac{1}{\sqrt{n_{i-1}}},$$

and then notice that the product in the σ_i expression above simply becomes one. Similarly, if we choose to keep the gradient magnitudes constant, we would choose

$$\sigma_{W_i} = \sqrt{2} \frac{1}{\sqrt{n_i}}.$$

In both cases, it is worth noting that only the input or output size of a layer affects the standard deviation of the initialization distrubition. If we want to keep the outputs constant, we use n_{i-1}, which is the number of inputs to the layer, and if we want to keep the gradients constant we use n_i, which is the number of outputs.

The downside of Kaiming initialization is that whichever value we *don't* choose can vary quite a bit between layers. For example, if we choose to keep the output magnitudes constant, the gradients may change dramatically between layers. To solve that problem, Xavier initialization attempts to balance both the gradient and the output magnitudes by using the average of the input and output sizes. That is, in Xavier initialization

$$\sigma_{W_i} = \sqrt{2}\sqrt{\frac{2}{n_{i-1} + n_i}}.$$

In this case, neither the outputs not the gradients have truly constant magnitude, but we also prevent the variance on either from getting too high. Most of the time, in practice, Xavier initialization works better than Kaiming initialization.